CS 7545: Machine Learning Theory **Fall 2021**

Lecture 11

Name: Tyler LaBonte **Professor:** Santosh S. Vempala

1 Learning Halfspaces

Suppose our data domain is Euclidean $\Omega = \mathbb{R}^d$ or boolean $\Omega = \{0,1\}^d$. An important object is the halfspace, or linear threshold function. For $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$,

$$H(\boldsymbol{w}, b) = \{\boldsymbol{x} \in \Omega : \boldsymbol{w}^\top \boldsymbol{x} \geq b\}. \tag{1}$$

Many seemingly unrelated problem classes can be written as halfspaces, so they are pretty powerful. Let's see some examples. Suppose $\boldsymbol{x} = (x_1, x_2, \dots, x_k)$ is an ordered set of booleans. Then a conjunction

$$x_1 \wedge \bar{x_3} \wedge \cdots \wedge x_k \tag{2}$$

with r negated terms may be written as the halfspace

$$\boldsymbol{w}^\top = (1, 0, 1, \dots, 1) \quad b = k - r. \tag{3}$$

Similarly, a disjunction

$$x_1 \vee \bar{x_3} \vee \cdots \vee x_k \tag{4}$$

with r negated terms may be written as the halfspace

$$\boldsymbol{w}^\top = (1, 0, 1, \dots, 1) \quad b = 1 - r. \tag{5}$$

One surprising equivalence is that of decision lists. The decision list

$$\text{if } x_1 = 1 \text{ then 1 elif } x_2 = 0 \text{ then 0 elif} \dots \text{elif } x_k = 0 \text{ then 0 else 1} \tag{6}$$

may be written as the halfspace

$$2^k x_1 - 2^{k-1}(1 - x_2) + \cdots + 2x_k + 1 \geq 1. \tag{7}$$

The exponentially decreasing weights ensure that the order of the decision list is preserved.

Note that, while there are infinitely many halfspaces in $\mathbb{R}^d$, they are still feasible to learn. To get an intuition for this, consider $\Omega = \{0,1\}^d$. Say that halfspaces are distinct if they assign a different label to at least one element of the domain. Then, there are $\binom{2^d}{d} \leq 2^{d^2}$ distinct halfspaces, which has a very reasonable logarithm of d^2. We will formalize this idea as VC-dimension in a

future lecture.

2 Perceptron Algorithm

Suppose we are in the mistake bound model and we want to learn the optimal halfspace $\boldsymbol{w}^\star$. We can assume $b = 0$, since we can always append a 1 to $\boldsymbol{w}^\star$ and a -1 to $\boldsymbol{x}$, thus working in $\mathbb{R}^{d+1}$ with the same effect. We can also assume $\|\boldsymbol{w}^\star\|_2 = 1$ and $\|\boldsymbol{x}\| \leq 1$.

Define the label of a point in terms of the optimal halfspace: $\ell(\boldsymbol{x}) = \operatorname{sgn}(\langle \boldsymbol{w}^\star, \boldsymbol{x}\rangle)$. Then, we can learn a halfspace using the perceptron algorithm:

1. Initialize $\boldsymbol{w} = \mathbf{0}$.
2. On example $\boldsymbol{x}$, predict $\operatorname{sgn}(\langle \boldsymbol{w}, \boldsymbol{x}\rangle)$.
3. If mistake, update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{x}\ell(\boldsymbol{x})$.

Intuitively, well-separated datasets should be more conducive to linear classifiers. This is quantified by the *margin*: $\gamma = \min_{\boldsymbol{x}} |\langle \boldsymbol{w}^\star, \boldsymbol{x}\rangle|$.

Theorem 2.1: Perceptron Mistake Bound

Suppose there exists $\boldsymbol{w}^\star$ with $\gamma > 0$. Then, the number of mistakes made by the perceptron algorithm is at most $\frac{1}{\gamma^2}$.

Proof: Consider the potential function

$$\cos(\boldsymbol{w}, \boldsymbol{w}^\star) = \frac{\langle \boldsymbol{w}, \boldsymbol{w}^\star\rangle}{\|\boldsymbol{w}\|}. \tag{8}$$

It starts out at zero. After one mistake,

$$\langle \boldsymbol{w}, \boldsymbol{w}^\star\rangle \leftarrow \langle \boldsymbol{w} + \boldsymbol{x}\ell(\boldsymbol{x}), \boldsymbol{w}^\star\rangle = \langle \boldsymbol{w}, \boldsymbol{w}^\star\rangle + \ell(\boldsymbol{x})\langle \boldsymbol{w}^\star, \boldsymbol{x}\rangle. \tag{9}$$

Since $\ell(\boldsymbol{x})$ and $\langle \boldsymbol{w}^\star, \boldsymbol{x}\rangle$ have the same sign, $\langle \boldsymbol{w}, \boldsymbol{w}^\star\rangle$ increases by at least γ for each mistake. So, after t mistakes, it increases by at least γt. For the denominator,

$$\langle \boldsymbol{w}, \boldsymbol{w}\rangle \leftarrow \langle \boldsymbol{w} + \boldsymbol{x}\ell(\boldsymbol{x}), \boldsymbol{w} + \boldsymbol{x}\ell(\boldsymbol{x})\rangle \tag{10}$$

$$= \langle \boldsymbol{w}, \boldsymbol{w}\rangle + \langle \boldsymbol{x}, \boldsymbol{x}\rangle + 2\ell(\boldsymbol{x})\langle \boldsymbol{w}, \boldsymbol{x}\rangle. \tag{11}$$

The middle term is at most 1 by definition and the last term is negative because we made a mistake. Thus, it increases by at most 1 each mistake, and at most t after t mistakes. So,

$$\cos(\boldsymbol{w}, \boldsymbol{w}^\star) \geq \frac{\gamma t}{\sqrt{t}} = \gamma\sqrt{t}. \tag{12}$$

If $\gamma\sqrt{t} \le 1$, it must be that $t \le \frac{1}{\gamma^2}$. ■

There is a generalization of this theorem with relaxed assumptions that obtains

$$t \le \frac{\|\boldsymbol{w}^\star\| \max \|\boldsymbol{x}\|}{\gamma^2}. \tag{13}$$

However, γ can be exponentially small even on the boolean cube, so it is not completely efficient. To solve this, there is a modified perceptron which makes $\frac{\log n}{\sigma^2}$ mistakes and outputs a halfspace that answers correctly when the margin is above σ, and has no guarantees otherwise. Another way to solve this is by using linear programming.

3 The Kernel Trick

What if the threshold is not linear? Well, we can project to a higher-dimensional space where it is linear. For example, suppose the threshold is a quadratic,

$$\boldsymbol{x}^\top \mathbf{A} \boldsymbol{x} \ge 0 \implies \sum_{i,j} a_{ij} x_i x_j \ge 0. \tag{14}$$

Then, we let $y_{ij} \coloneqq x_i x_j$ to obtain the linear threshold $\sum_{i,j} a_{ij} y_{ij} \ge 0$. We now have $\binom{d}{2}$ dimensions, but this is fine because perceptron gives a dimension-free guarantee!

Definition 3.1: Kernel

$K : \Omega \times \Omega \to \mathbb{R}$, $K(\cdot,\cdot)$ is a legal kernel if $\exists \phi : \mathbb{R}^d \to \mathbb{R}^m$ such that $K(\boldsymbol{x},\boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle$. This is equivalent to the matrix $\mathbf{K}$ being positive semidefinite.

Kernels help us compute inner products in projected high-dimensional spaces without actually knowing the mapping (which can be inefficient or have no closed form). Some kernels include $\langle \boldsymbol{x},\boldsymbol{y}\rangle^k$ and $\exp(-\|\boldsymbol{x}-\boldsymbol{y}\|^2)$.

To define the kernel perceptron, we first observe that in the perceptron algorithm, we can keep track of the set of mistakes instead of updating one at a time: $\boldsymbol{w} = \sum_i \boldsymbol{x}_i \ell(\boldsymbol{x}_i)$ and prediction $\langle \boldsymbol{w}, \boldsymbol{x}\rangle = \sum_i \langle \boldsymbol{x}_i, \boldsymbol{x}\rangle \ell(\boldsymbol{x}_i)$. For kernels, we would have $\boldsymbol{w} = \sum_i \phi(\boldsymbol{x}_i)\ell(\boldsymbol{x}_i)$, but we don't want to actually compute ϕ. Luckily, we can use the kernel:

$$\langle \boldsymbol{w}, \phi(\boldsymbol{x})\rangle = \sum_i \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x})\rangle \ell(\boldsymbol{x}_i) \tag{15}$$

$$= \sum_i K(\boldsymbol{x}_i, \boldsymbol{x}) \ell(\boldsymbol{x}_i). \tag{16}$$

Thus, we can learn a halfspace in $\mathbb{R}^m$ without actually computing the mapping ϕ.